Reinforcement Learning Based SDN Controller Load Balancing

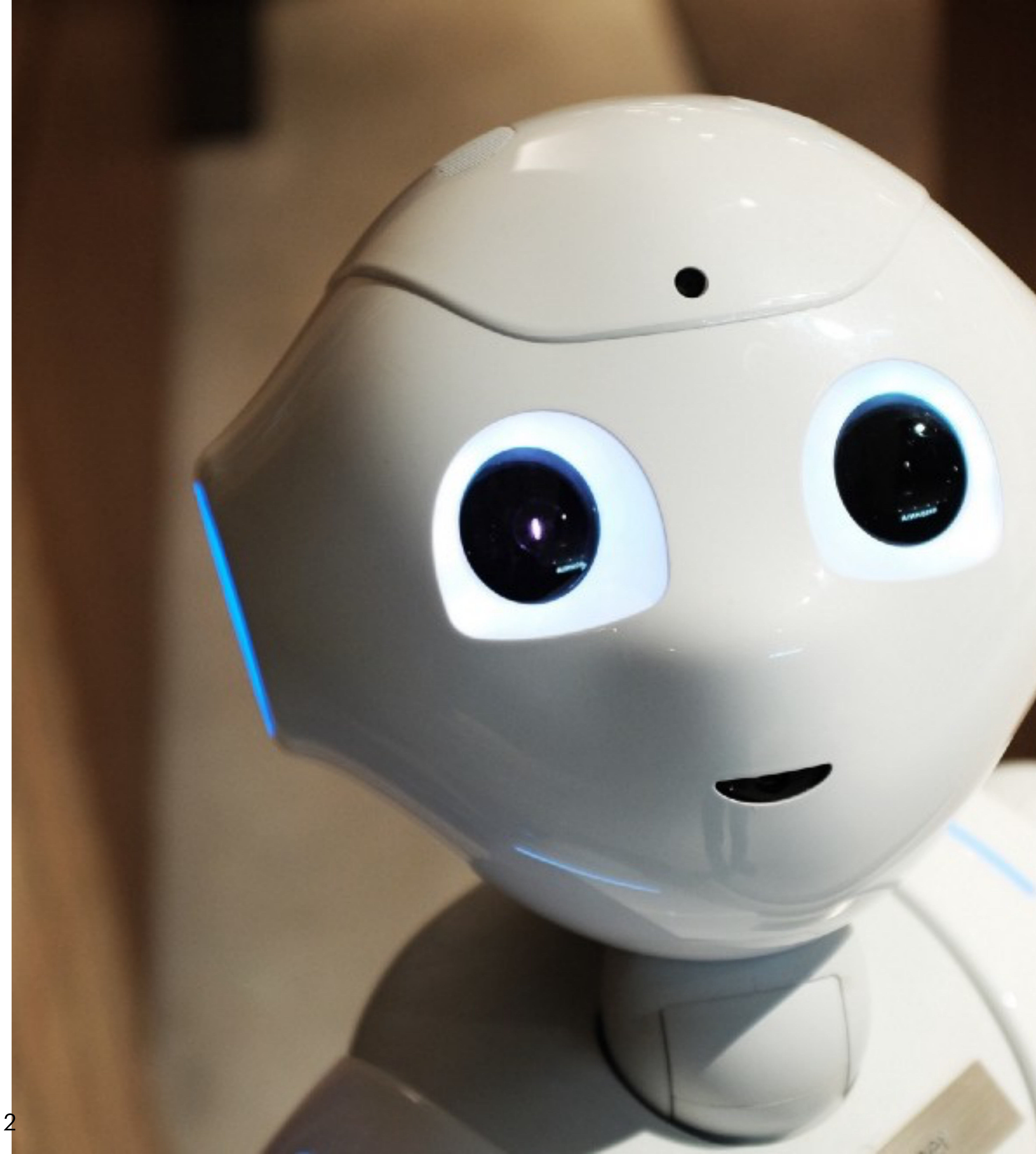
BTP Presentation - MA 499

Indian Institute of Technology, Guwahati

AB Satyaprakash - 28 April, 2022

Presentation Outline

- Introduction
- Load Balancing Problem
- Important terminologies
- System Model
- Problem Statement
- Implementation
- Experimental Setup
- Future Work
- Bonus Work

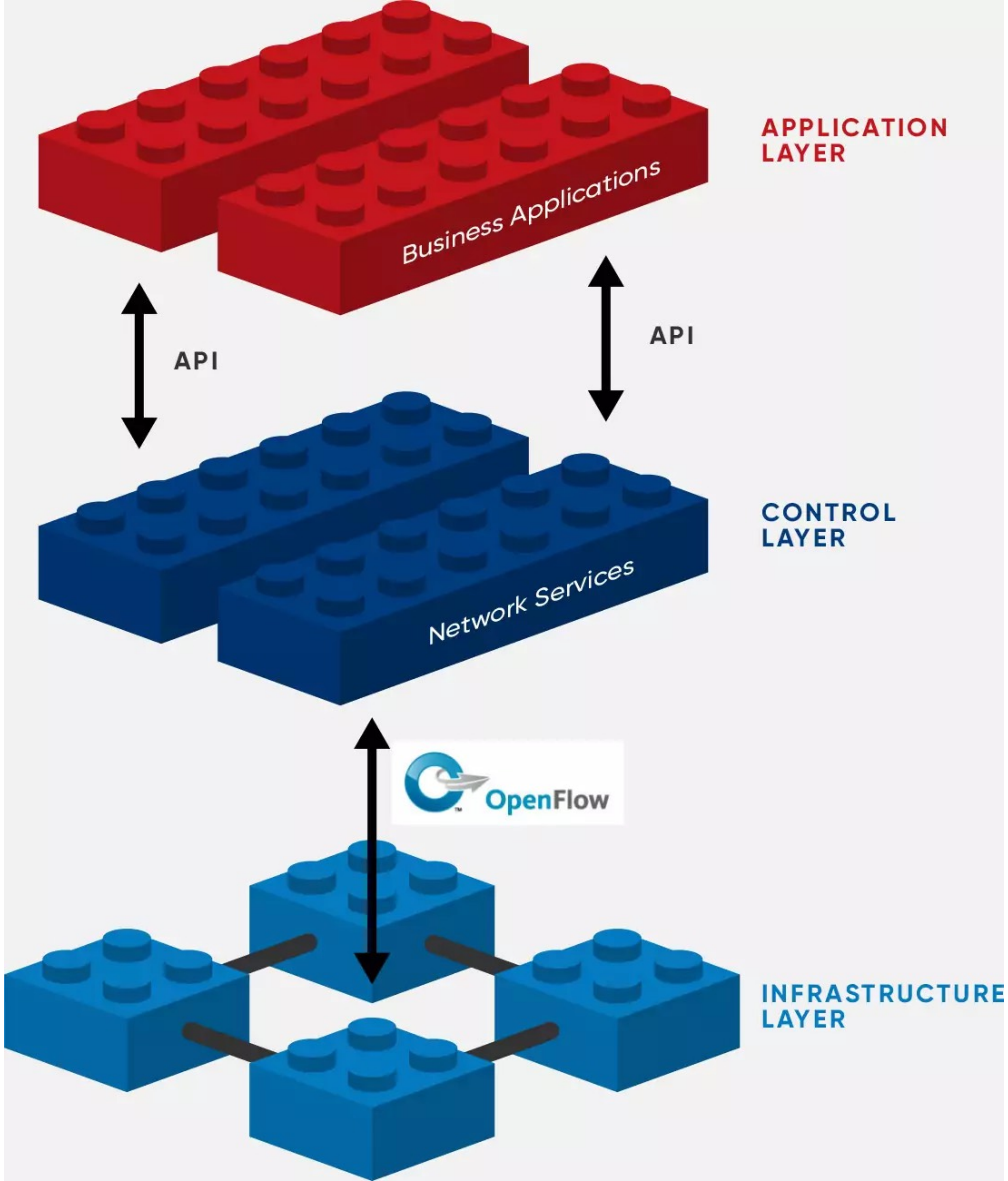


Introduction

SDN and Q-Learning

What is Software Defined Networking?

- Software-defined networking (SDN) separates the control plane and data plane of network devices.
- Unique advantages - centralised control, network programmability.
- Distributed vs Concentrated control. Northbound, Southbound, EastWest APIs.



What is Q-Learning?

- Q-learning is an off policy reinforcement learning algorithm that seeks to find the best action to take given the current state.
- Off-policy means it can take random action to maximise total reward.
- Model-free means it does not require a model of the environment, and thus can handle problems with stochastic transitions.
- The optimal behaviour to maximise reward is learned through experiences with the environment and observations of how it reacts.

The Problem

SDN Controller Load Balancing

SDN Controller Load Balancing

- Distributed deployment using multiple controllers -
  - improves scalability of the control plane
  - avoids single-point failures in centralised deployment
- Load on controllers primarily arises due to no flow entry for a newly arrived packet or communication between controllers.
- Any load balancing mechanism should ensure - proper load distribution, no migration conflicts, no creation of new overloads.

- In this figure, we see 3 controllers, of which C3 is overloaded, while C1 and C2 are normal.
- Migration of switch S5 and S9 to C1 and C2 respectively reduces the overload on C3.

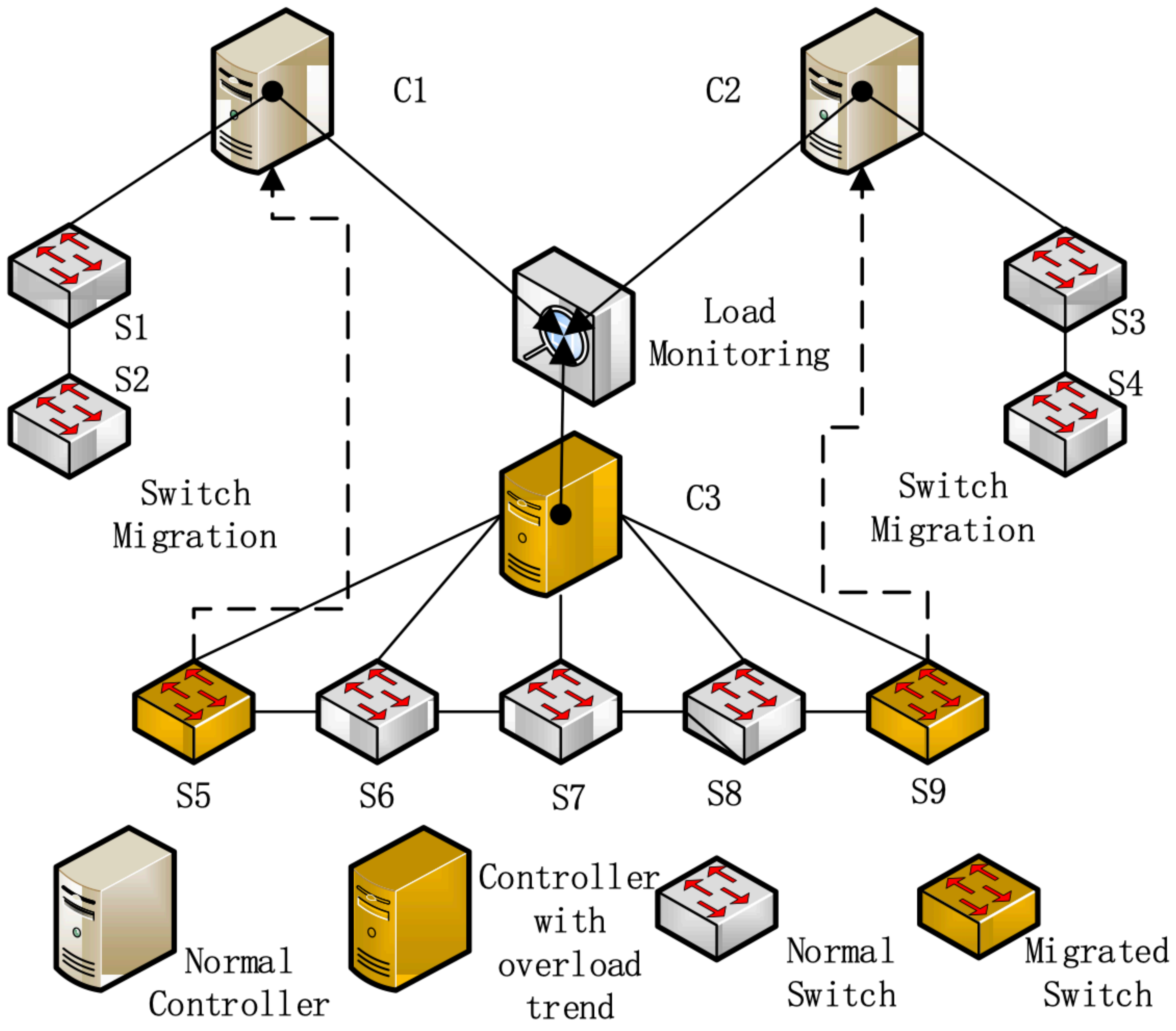


Fig. Controller load diagram showing switch migration

Switch Migration Design

Important terminologies

Switch Migration Design

Controller's Load	Controller's Load Capacity	Controller's Load Ratio	Discrete Coefficient
The load on a controller is the sum of the packet-in messaging rates on it's switches.	Controller's can have varying load capacities depending on CPU performance, number of processors, memory size, etc.	The load ratio of a controller is the ratio of controller's load to its load capacity.	Discrete coefficient of a system of controller's is the deviation of controllers' load ratios from the mean load ratio.
$L_{C_i} = \sum_{k=1}^{n} L_{S_k}$	$C_{C_i} = f(CPU, Memory)$	$R_{C_i} = \frac{L_{C_i}}{C_{C_i}}$	$D = \frac{\left(\sqrt{\sum_{i=1}^{n} \left(R_{C_k} - \overline{R}\right)^2 / 2}\right)}{\overline{R}}$

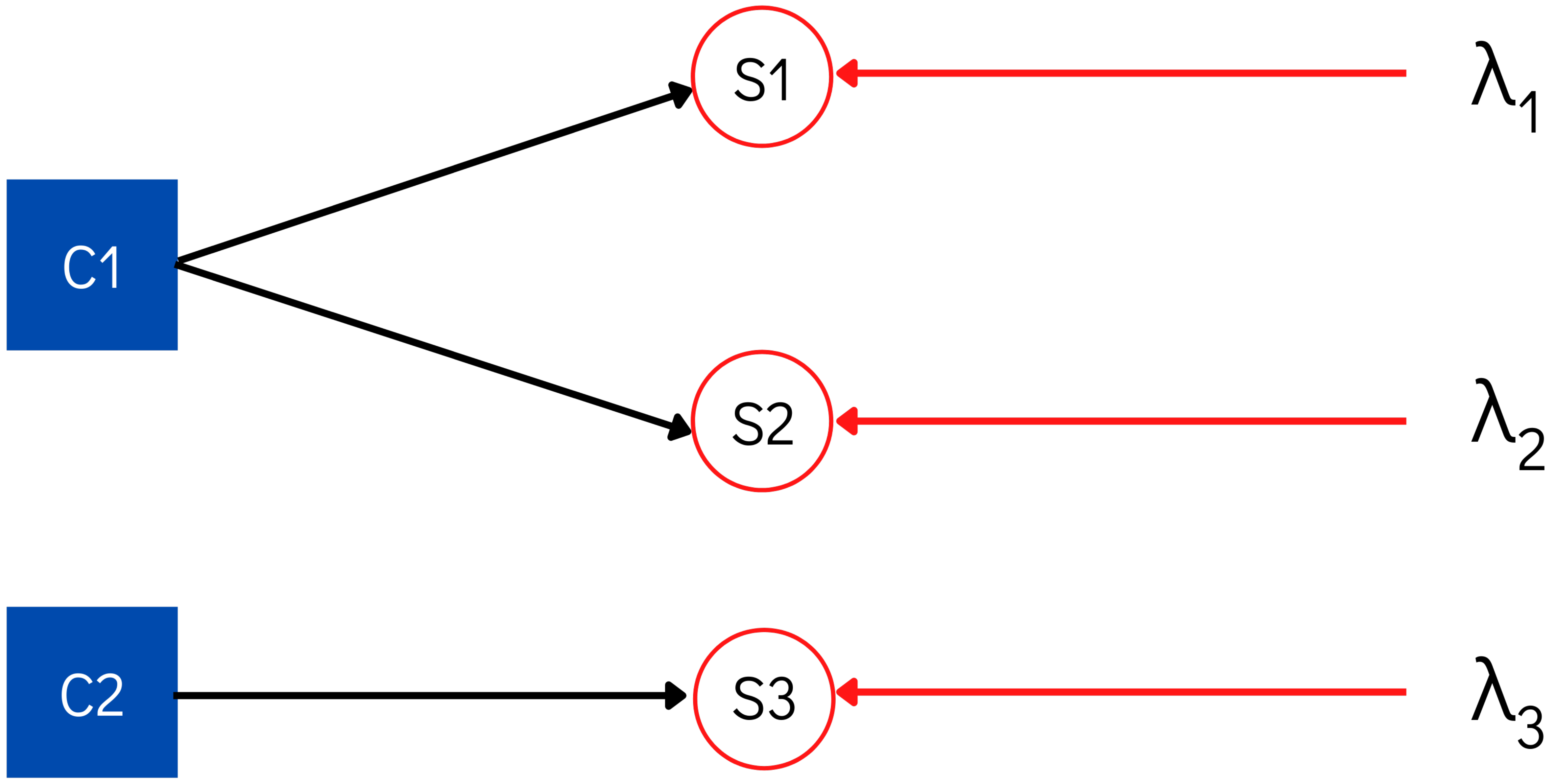
Fig. Table showing different terminologies associated with Controller load balancing

System Model

States, Actions, Rewards, Costs, Assumptions …

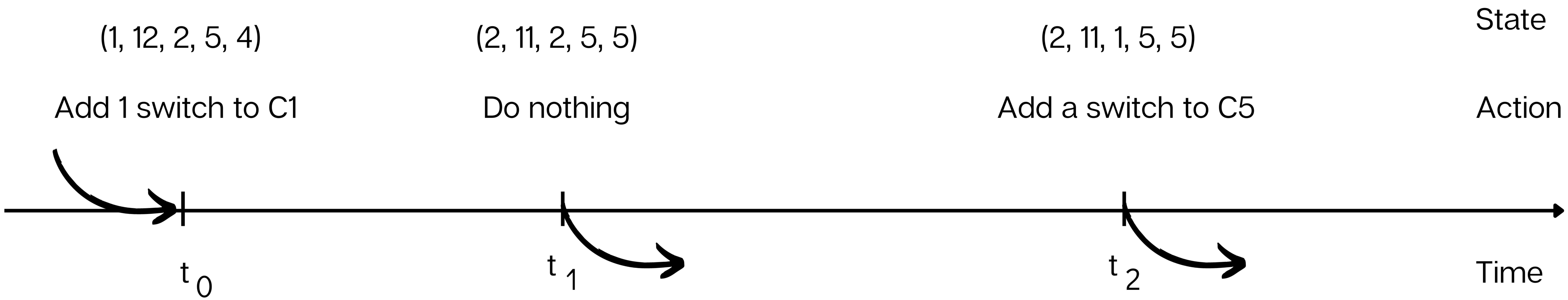
Assumptions

- Flows on switch modelled by Poisson process with mean λ.
- Service times modelled by Exponential process with mean 1/μ.
- No queues, infinite servers, no waiting times.
- Arrival of a new flow or departure of serviced flow is taken as a decision epoch.



State and Action Spaces

- State space is a set of tuples (states) consisting of number of flows on each controller.
- For example, in a k controller setup, any state would look like (number of flows in C_1, number of flows in C_2, ..., number of flows in C_k)
- Action space is a set of 3 actions = {do nothing, add a switch to C_i, remove a switch from C_i}
- For each (s, a) pair, we can obtain the next state s′ to which the system will move into with a probability $p_{ss'}(a)$.



- In this figure we see state transitions at some random consecutive time points - t_0, t_1 and t_2 for system with 5 controllers.
- For each state, the new state comes after an action is taken.
- The state is also modified at each time step (decision epoch) when a flow arrives or departs.

Reward and Cost

- Reward is a key component for Q-Learning. It has to be a fairness metric that reflects how favourable the action was.
- Reward is taken as negative of discrete coefficient, i.e, $r(s, a) = -D_{ij}$
- Every action incurs some overhead due to switch migration, since flows are redirected to a new controller.
- The cost in our case is defined as:
  - 0 if action is *do nothing*
  - Number of flows migrated if action is *switch migration*

Problem Formulation

Markov Decision Process

Markov Decision Process

- MDP is a discrete-time stochastic control process. Framework for modelling decision making when outcomes are partly random and partly based on specific actions.
- MDP consists of state and action spaces, rewards, models and policies.
- In our case we need a CMDP (Constrained Markov Decision Process), since we have to optimise reward and constrain cost incurred.

Problem Formulation

- Objective: Maximise the total discounted reward over the infinite horizon subject to constraints on total discounted cost over the infinite horizon.
- Let S_{max} is the constraint on average number of switch exchanges between controllers, the formulated constrained MDP can be given as:
- $\max_{\pi \in U} \lim_{T \to \infty} \left(\sum_{t=0}^{T-1} \alpha^t E_\pi[r(s_t, a_t)] \right)$, where $\lim_{T \to \infty} \left(\sum_{t=0}^{T-1} \alpha^t E_\pi[c(s_t, a_t)] \right) \leq S_{max}$

Implementation

Random, Greedy and Q-Learning approaches

Random and Greedy approaches

- In random approach controllers are classified into O_{domain} and I_{domain} representing overloaded and underloaded domains respectively.
- A random switch S_i is then transferred from controllers in O_{domain} to I_{domain} at every time step.

- In greedy approach we decide upon a set of migration triplets by preprocessing the data at every time step.
- Triplet = (C_{out}, C_{in}, S_i)
- S_i is such that load ratio is minimised and efficiency is maximised. Then at every time step, we migrate S_i from C_{out} to C_{in}.

Q-Learning implementation

- We use an online Q-Learning implementation with ε-greedy strategy to ensure balance exploration and exploitation. Q-values are updated based on the following equation -
- $Q_{n+1}(s,a) = (1 - a(n))Q_n(s,a) + a(n)\left[r(s,a) - l_n c(s,a) + \gamma \max a' Q_n(s', a')\right]$
- Here l_n is a Lagrange multiplier which should be iterated along the timescale of b(n) like $l_{n+1} = A[l_n + b(n)(S_n - S_{max})]$.
- A is a projection operator to keep the Lagrange multiplier between [0, L] for a large L. S_n is given as a running sum of α raised to the power of iteration, i.e.,$S_n = 1.\alpha^0 + 1.\alpha + 0.\alpha^2 + \dots$

Q-Learning implementation

Symbols	Remarks	Values
Learning Rate - a(n) or $\boldsymbol{\alpha}$	The sequence a(n) satisfies $\sum_{n=1}^{\infty} a(n) = \infty; \sum_{n=1}^{\infty} (a(n))^2 < \infty$	$\frac{1}{(n+1)^{0.6}}$
Discount factor - γ	This is essential in determining the point of convergence	0.99
Epsilon in ε-greedy	Here ε is decremented from an initial value to a ε-min	ε-initial = 0.7 ε-dec = 0.0025 ε-min = 0.05
LM Sequence - b(n)	The sequence b(n) satisfies $\sum_{n=1}^{\infty} (a(n) + b(n))^2 < \infty; \lim_{n\to\infty} \frac{b(n)}{a(n)} \to 0$	$\frac{1}{n}$
S_{max}	Maximum number of switch exchanges possible	4000
LM maximum value - L	A is a projection operator to keep the LM in range [0, L]	10000

Algorithm 3: Load Balancing using Q-learning

input : C: `set of all controllers`
$C_{cluster}$: `initial arrangement of switches and controllers`
$load_{array}$: `load on switches for any iteration i`
$n(s)$: `number of switches in the SDN environment`
$n(C)$: `number of controllers in SDN environment`

output: cumulative discounted rewards
discrete coefficients
number of switch exchanges between controllers

begin

- Initialize the evaluation matrix Q with all 0, γ, α, $b(n)$, ε, S_{max}, S_n, and l_0.
- **foreach** *timesteps* **do**
    - Determine the current system state (using last state and flows)
    - **if** *exploration phase* **then**
        - Choose one of the feasible actions at random
    - **else**
        - action = arg $\max_a$Q(s,a)
    - Observe reward r(s,a; $l_n) = r(s,a) - l_n c(s,a)$
    - Go to next state s'
    - Update **Q(s,a)** according to equation 6.5
    - Update the LM according to equation 6.6
    - Set current state to next state (s $\leftarrow s'$)
    - Record and update parameters for plotting graphs
    - **if** $\varepsilon > \varepsilon$-*min* **then**
        - $\varepsilon = \varepsilon * \varepsilon$-dec
    - **else**
        - $\varepsilon = \varepsilon$-min

Experimental Setup

Data generation and example working

Data Generation

- An arrival rate Poisson λ means that the inter arrival times are exponentially distributed with a mean λ. The service rate is μ.
- For every switch we compute arrival times, and departure times based on the above information. Finally we form tuples of the following form:

tuple(timestamp, switch, flow +/-)

- Sorting these tuples we finally generate our data which looks like this.

Timestamp	Switch 01	Switch 02	Switch 03	...	Switch 34
0.2546	0	0	0	...	1
0.3947	1	0	0	...	0
...					
0.9649	0	0	1	...	0
0.9947	0	0	0	...	0

Simple Example

- Consider a system with 5 switches and 3 controllers. Given that the flows on switches are the tuples (2, 3, 2, 5, 4) and (2, 3, 3, 5, 4) at 2 consecutive time points t_i and t_{i+1}, let us observe a single iteration of Q-Learning.
- After iteration t_i let mappings be: C1 = {S1, S2}, C2 = {S4}, C3 = {S3, S5}.
- Current state = (5, 5, 6). New state (after incoming of flow) = (5, 5, 7). Choose action from ε-greedy strategy and a random controller (say C2).
  - Do nothing - next state = initial state = (5, 5, 7).
  - Add 1 switch - transfer most overloaded switch to C2. Next state = (5, 9, 3)
  - Remove 1 switch - transfer random switch from C2 to most underloaded controller. Next state = (10, 0, 7)

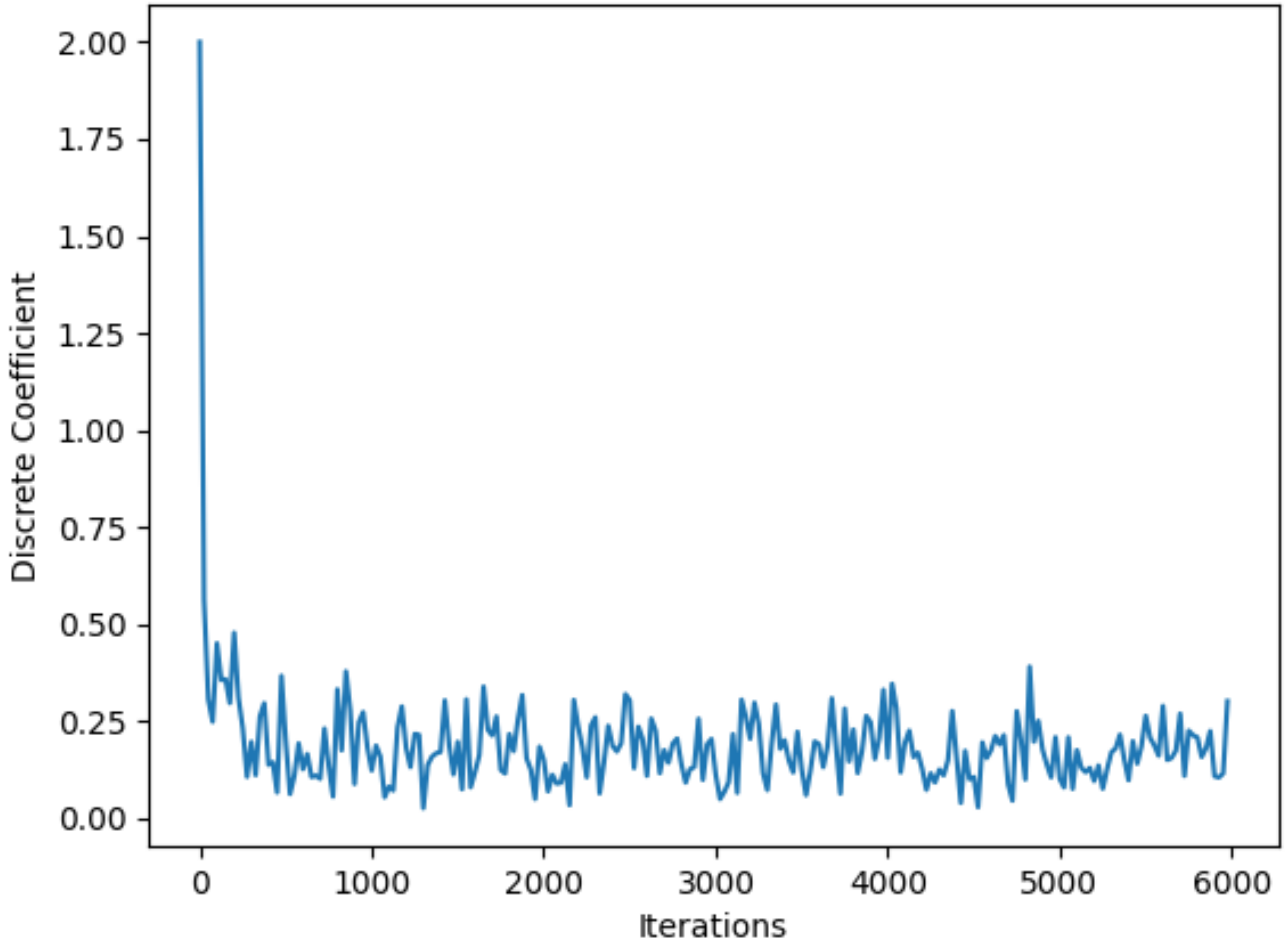
Simple Example

- Assuming a constant controller load capacity, we can obtain load ratio and D. Reward = -D.
- LM is updated according to the equation $l_{n+1} = A[l_n + b(n)(S_n - S_{max})]$
- Discounted reward = $reward * \gamma^n$. This is cumulated for every iteration.
- Finally ε value is discounted by ε-dec if it is more than ε-min. This process repeats again in next iteration.
- Finally plots are done for - discrete coefficient, cumulative discounted reward, and number of switch exchanges with time.
- Evaluation metric: Discrete coefficient and Cumulative discounted reward

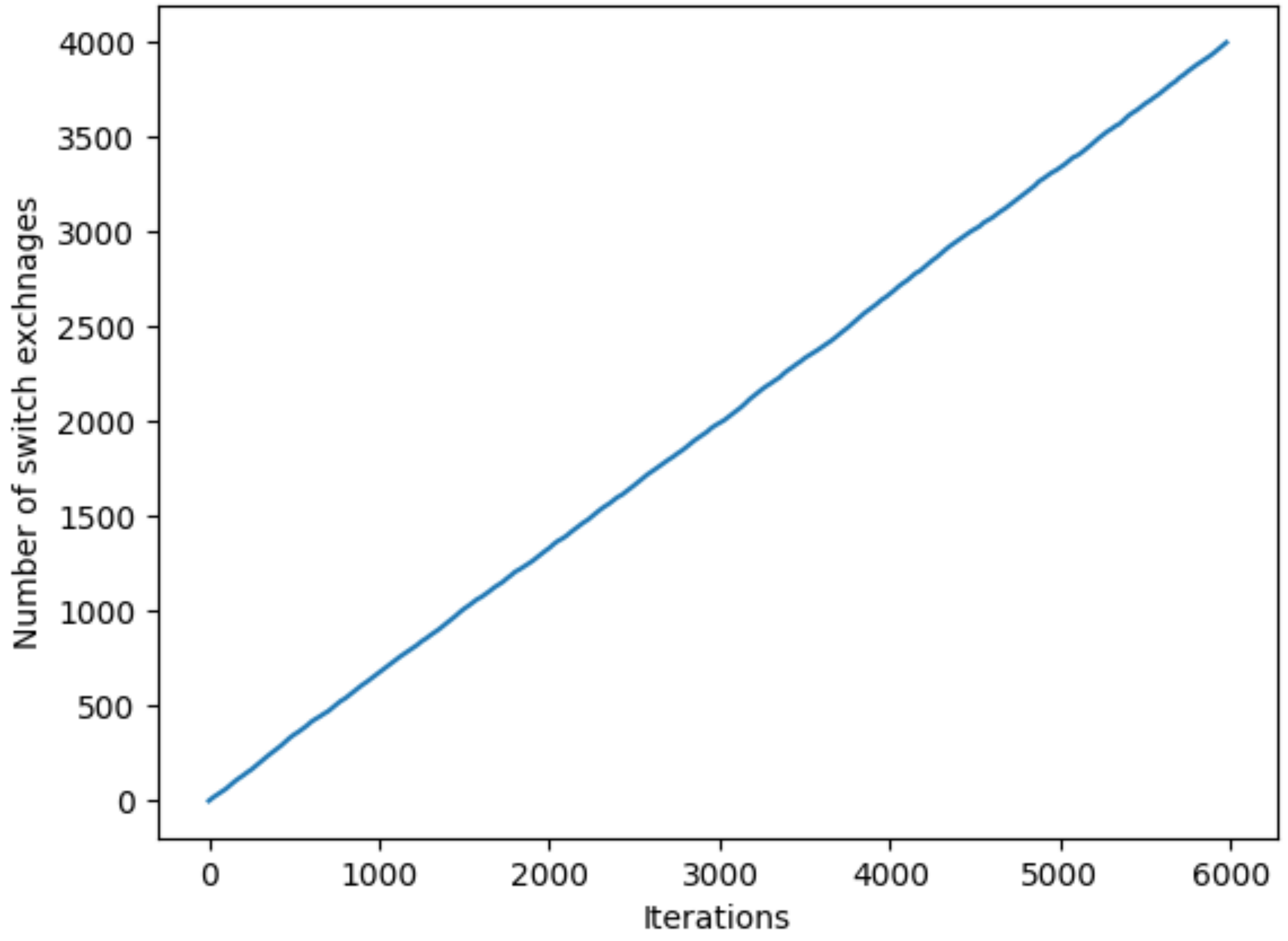
Experimental Setup

- We choose 3 scenarios - Heavy Load, Light Load, Skewed Load.
- Heavy Load - λ = random numbers in range (50, 100) & μ = a random number in range (1, 5)
- Light Load - λ = random numbers in range (1, 5) & μ = a random number in range (50, 100)
- Skewed Load - λ = high: random numbers in range (50, 100); low: random numbers in range (5, 10) & μ = a random number in range (25, 50)

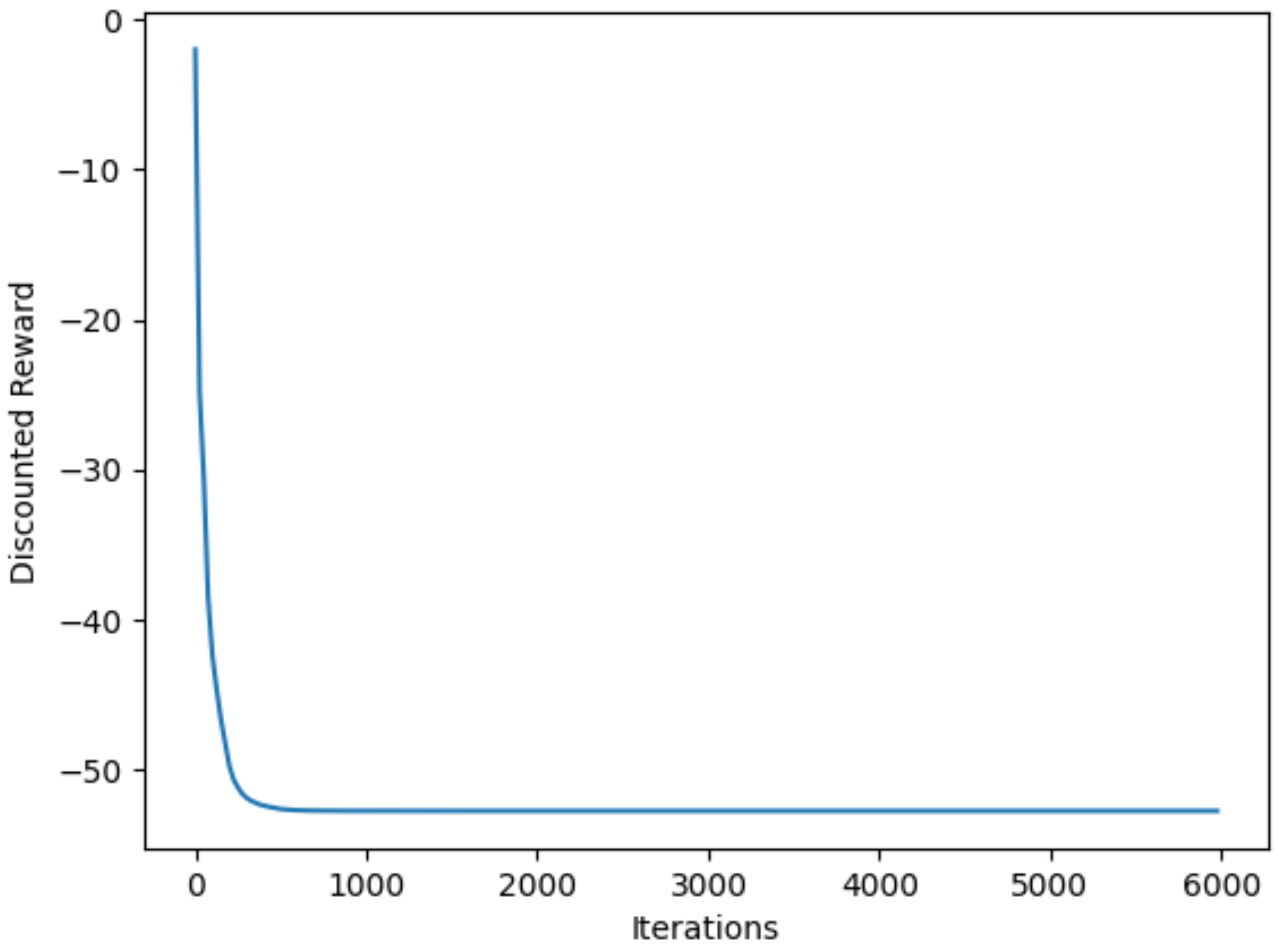
Load Heavy



Discrete Coefficient
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
1000
2000
3000
4000
5000
6000
Iterations

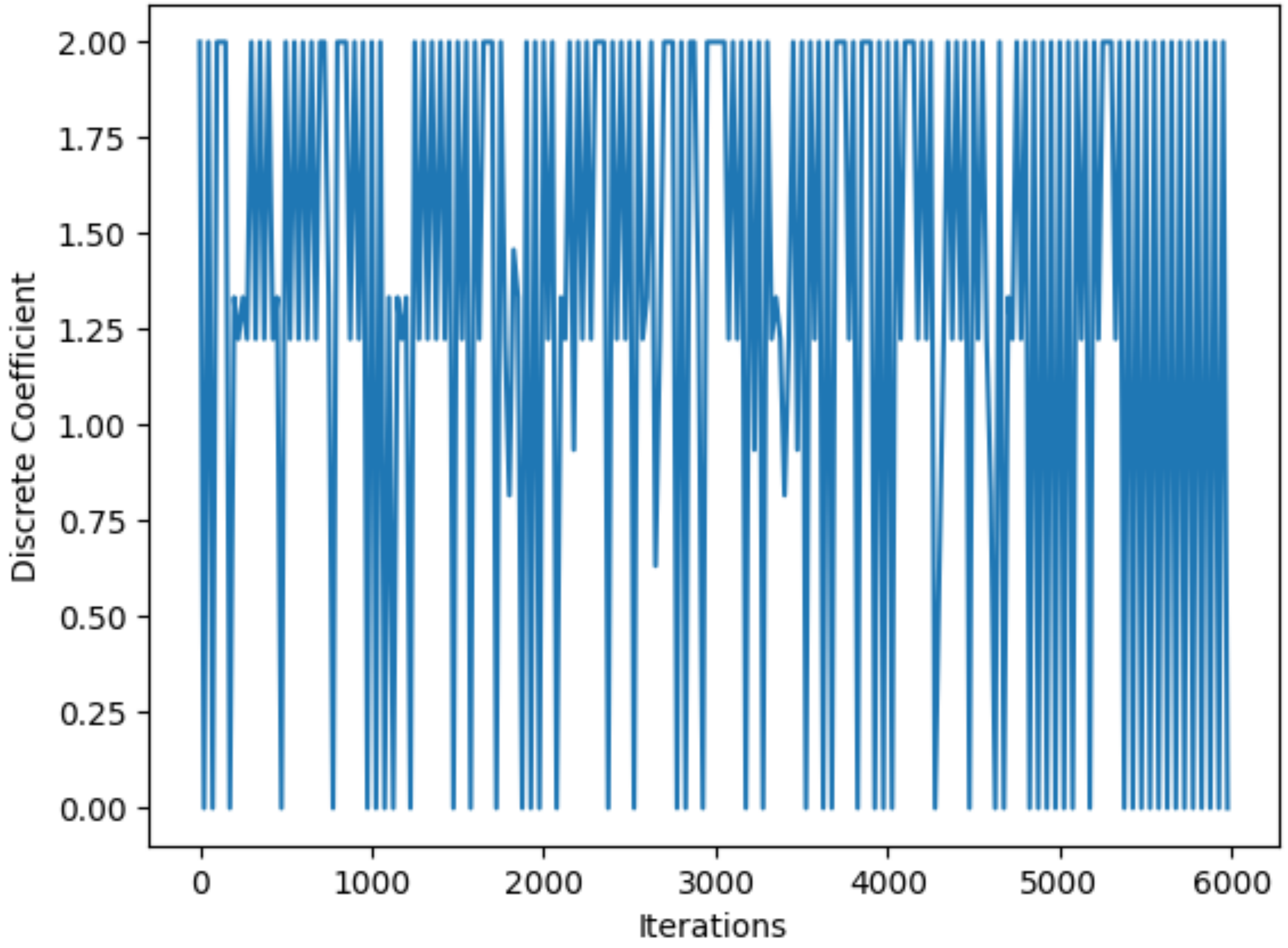


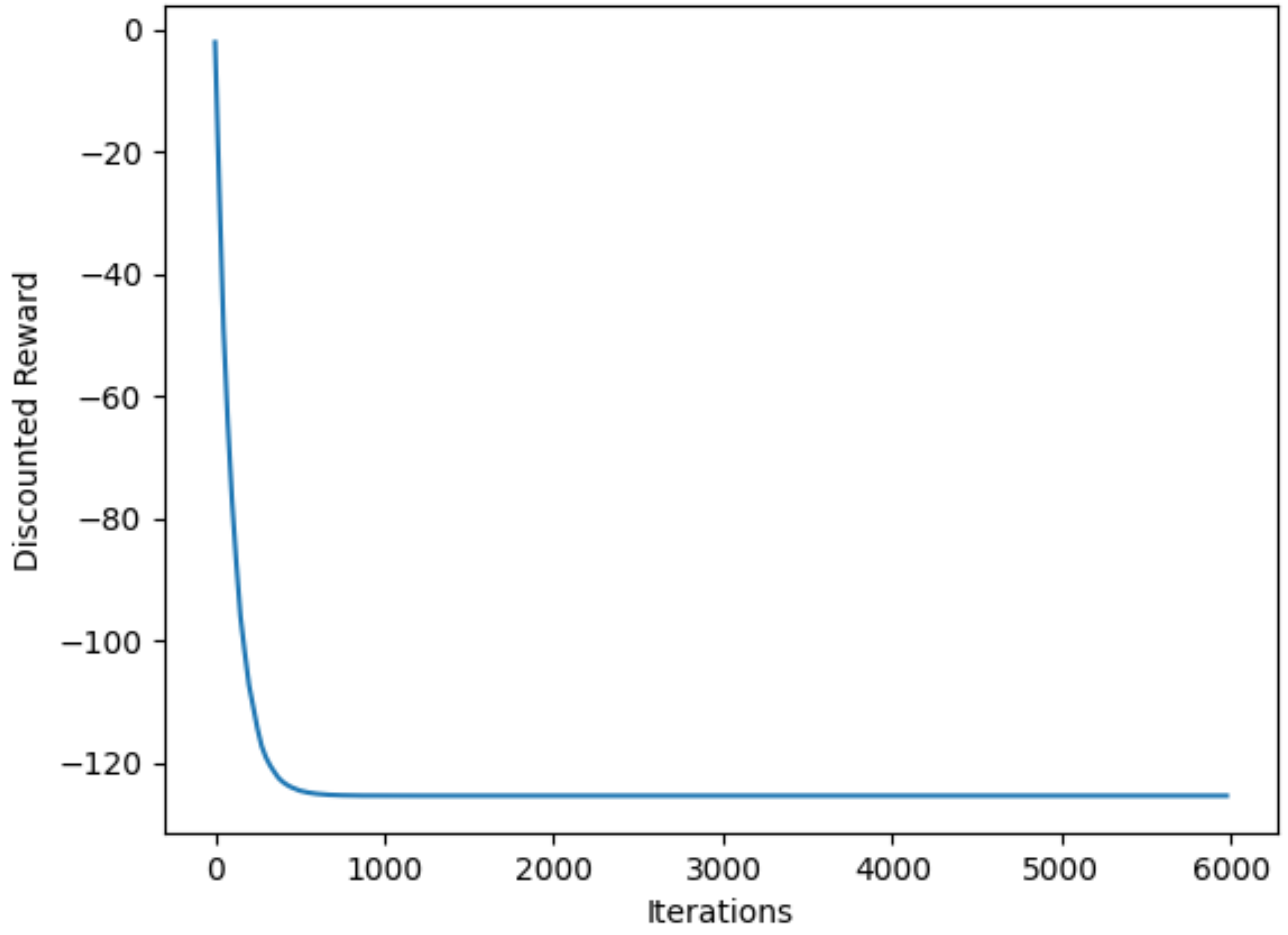
Number of switch exchnages
4000
3500
3000
2500
2000
1500
1000
500
0
0
1000
2000
3000
4000
5000
6000
Iterations

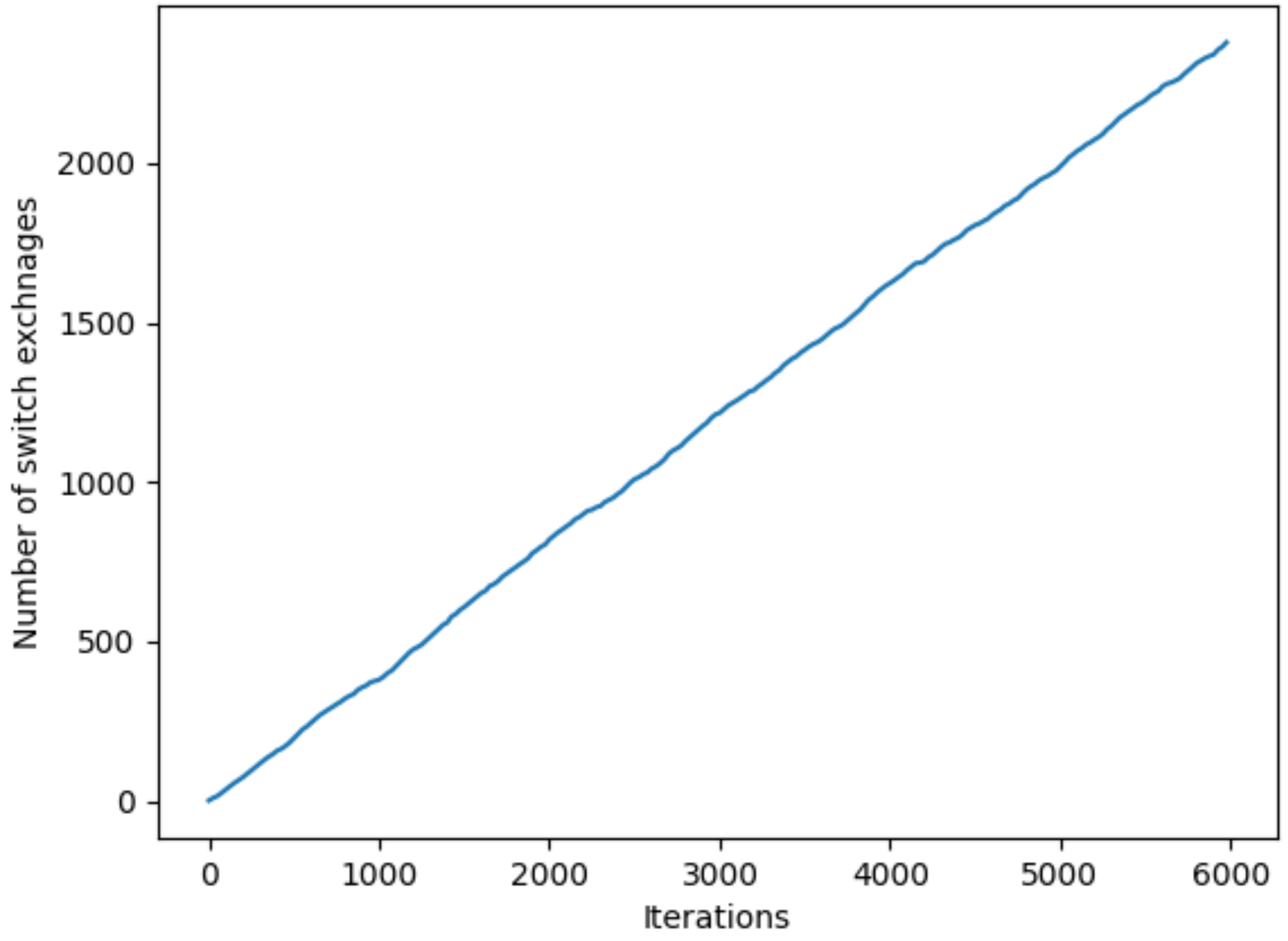


Discounted Reward
0
−10
−20
−30
−40
−50
0
1000
2000
3000
4000
5000
6000
Iterations

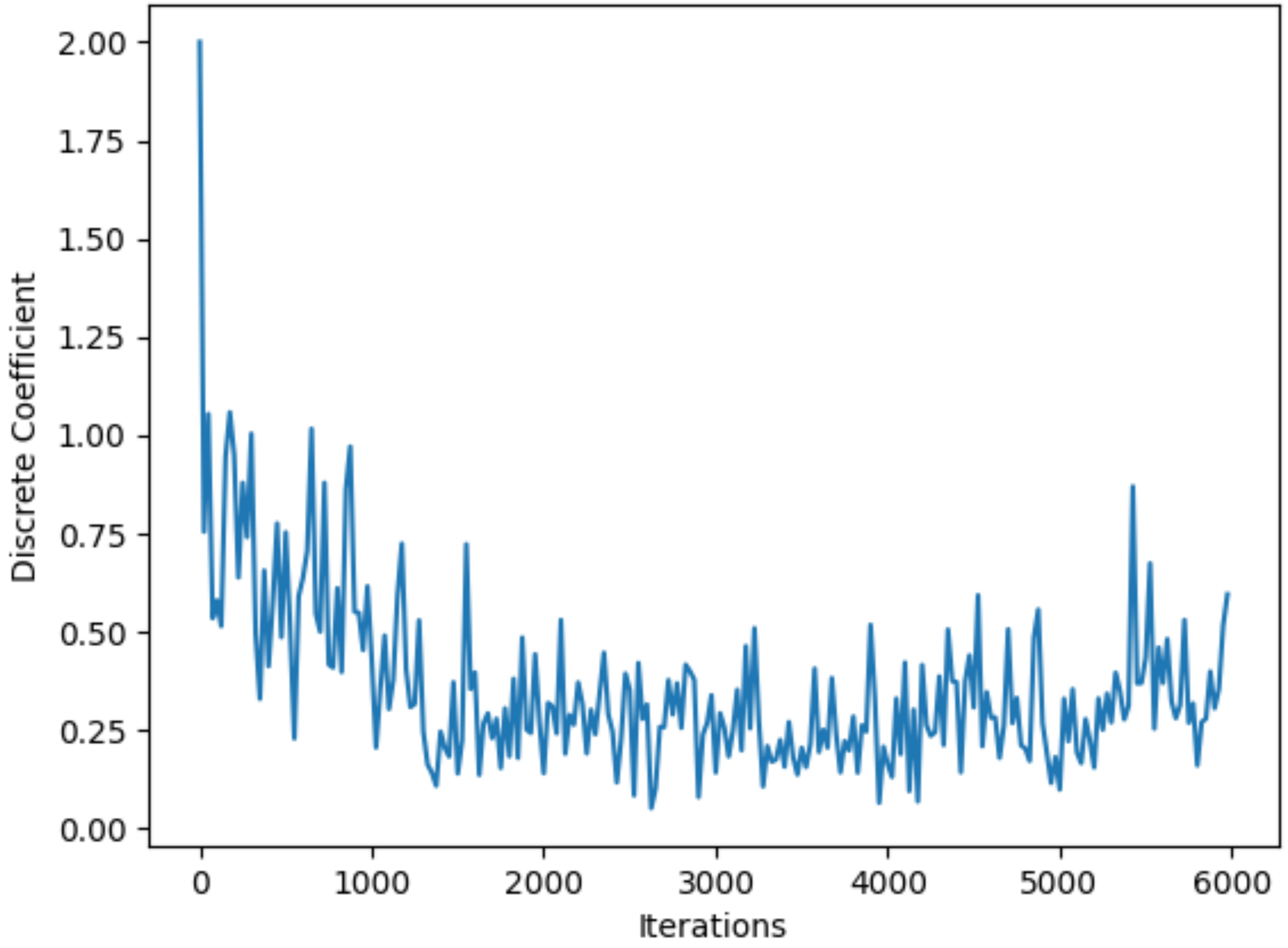
Load Light


Discrete Coefficient
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
1000
2000
3000
4000
5000
6000
Iterations

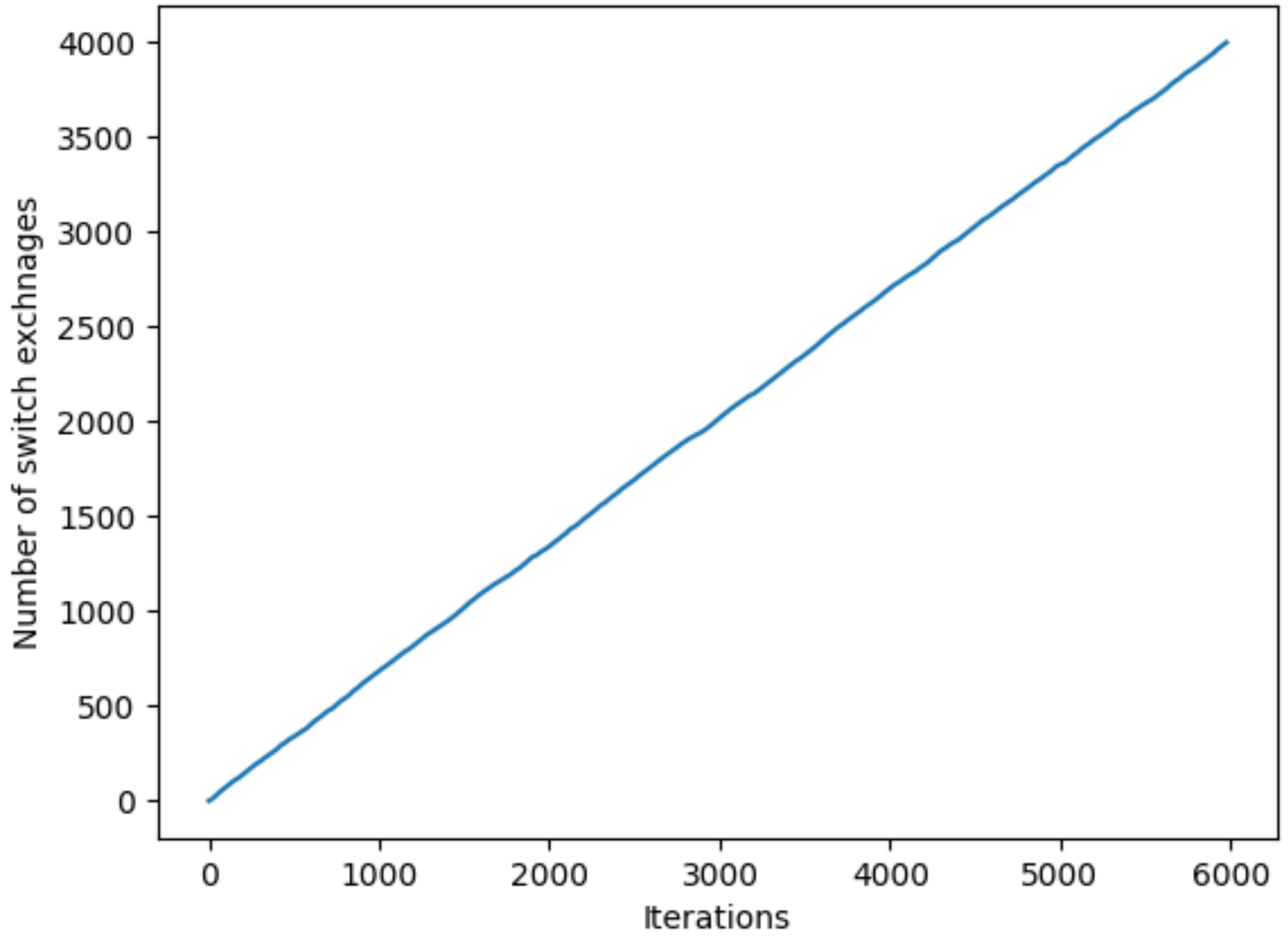

Discounted Reward
0
−20
−40
−60
−80
−100
−120
0
1000
2000
3000
4000
5000
6000
Iterations


Number of switch exchnages
2000
1500
1000
500
0
0
1000
2000
3000
4000
5000
6000
Iterations

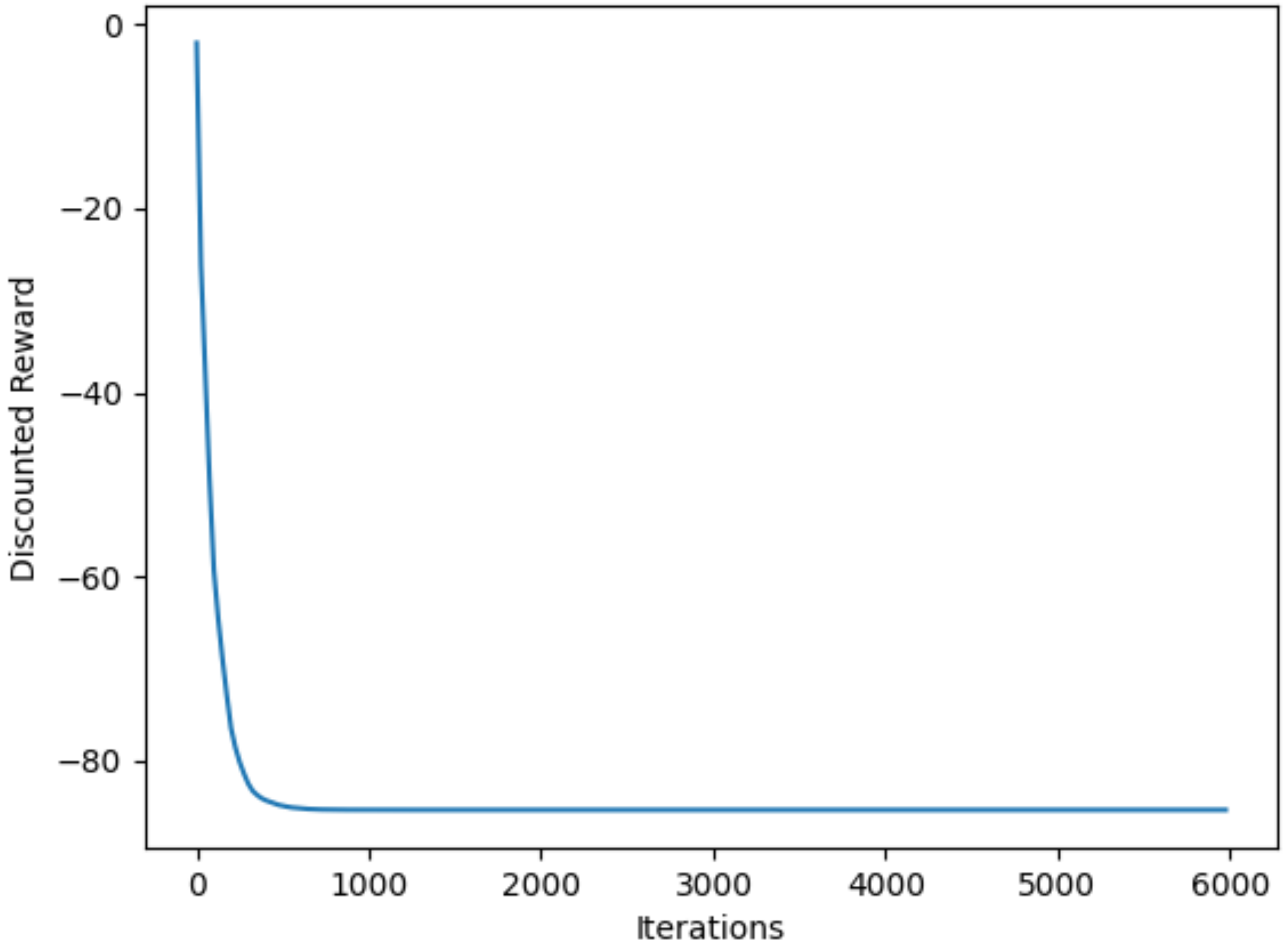
Load Skewed



Discrete Coefficient
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
1000
2000
3000
4000
5000
6000
Iterations



Number of switch exchnages
4000
3500
3000
2500
2000
1500
1000
500
0
0
1000
2000
3000
4000
5000
6000
Iterations



Discounted Reward
0
−20
−40
−60
−80
0
1000
2000
3000
4000
5000
6000
Iterations

Future Work

Real data, Deep Q-Learning

Future Work

- Our next goal would be work on real data obtained from different sources.
- Also we can explore other alternatives such as deep Q-learning to capture the network traffic behaviour to balance the controller and efficient switch migration considering dynamic networks.

Bonus Work

Greedy vs Random vs Q-Learning

- We observe that the discrete coefficient values are smallest in case of greedy approach than the other 2 approaches.